Mathematicaの自習用資料[1]

1　電卓的使い方の練習

- `2+2`とキーインし`Enter`キー[2]を押して下さい。
  待機カーソルが回転し、右の鍵かっこが２重になり、ウィンドウのタイトルが「Running...Untitle」となります。しばらく(数十秒)待つと、`Out[1]=4`という結果が表示されます。
- もう一度`2+2 [ Enter ]`をキーインして下さい。今度は`Out[2]=4`がすぐに得られます。
- かけ算：`2 3`
- べき：`2^100`
- 割算：`12345/15`
- `123/21`の答えは分数であらわされる。
- `N[%]`で上の分数の小数展開が得られる。% は直前の結果をあらわす。
- `N[123/21]`によって上の２つの操作が１度で行える。
- `N[1/37,100]`により、小数点以下１００位まで表示できる。
- `FactorInteger[1234567]`:1234567 の素因数分解.
- `N[Pi,1000]`:円周率を１０００桁まで表示。

2　補完機能

`FactorInteger`とキーインするかわりに、`Fac`とキーインしてからコントロールキーを押しながら`K`を押す。するとFacから始まる組み込み関数の一覧表が現れる。そこで`FactorInteger`をクリックするとFacのあとにtorIntegerが補われる。(FactorIntegerのどこまでをキーインするかは自由ですが余り短いと関数のリストが多すぎて選ぶのが大変です。またFactorIまでキーインすると関数が唯一つ決まるので一覧表は現れず自動的にntegerが補われます。)

---

[1] 最新版：‘‘http://fcs.math.sci.hokudai.ac.jp/doc/lect/cs99/mathematica.pdf’’
簡易マニュアル:”http://fcs.math.sci.hokudai.ac.jp/doc/lect/cs99/mathematica-manual.pdf”

[2] リターンキーではなく、テンキーの右下にあるキーです。

3　ノートブックのアウトライン機能

3.1　セルの選択・開閉

[3.1.1] 2+2 Enter キーを押して下さい。

[3.1.2] すると、まず右側に鍵括弧（これをセル括弧と呼ぶ）が現れ、それがすぐに2重になります。（これはそのセルの式を計算中であることを現す。）また、ウィンドウのタイトルが　Running...Untitle となることにも目を留めて下さい。

[3.1.3] しばらくたつと `In[1]:=2+2` と変わり、すぐに　`Out [1]:=4` という答が出ます。

[3.1.4] `In[1]=2+2` という行と `Out[1]=4` という行の右に各々短いセル括弧がついていて、さらにこの二つの括弧は長いセル括弧でくくられていますね。

[3.1.5] この大きなセル括弧にカーソルを持っていきクリックして下さい。するとそのセル括弧が黒くなります。（これをセルの選択という。）

[3.1.6] 同じ所でダブルクリックしてください。すると`Out[2]=4` の行が画面から消えると同時にセル括弧の下に小さい縦長四角が現れますね。これはセルが隠されていることを示します。

[3.1.7] もう一度ダブルクリックして下さい。すると隠れたセルが再現します。

3.2　節の作成

[3.2.1] `In[1]=2+2` の行の上あたりにカーソルを持って行き少し上下に動かして下さい。ある所でカーソルが横棒になります（水平Ｉ字カーソルという）。

[3.2.2] そこでクリックして下さい。すると横線が現れます（これを挿入横線と呼ぶ）。

[3.2.3] 1. とキーインして下さい。すると挿入横線が消えてその位置に 1. と点滅する縦棒が現れます。そこで続けて `Addition` とキーインして下さい。

[3.2.4] セルを選択してから（つまり右の鍵括弧の部分をマウスでクリックしてから）ALT+4 をとすると、`1.Addition` の活字が大きくなり、その前に■ 印が現れます。それと同時にその行のセル括弧と、大きなセル括弧が現れます。

[3.2.5] 現れた大きなセル括弧をダブルクリックして下さい。すると、見出しだけが残って後は隠れます。

[3.2.6] 再びダブルクリックして中身を表示しておいて下さい。

3.3　セルの自動的グループ化

[3.3.1] `Out[1]=4` の行の下にカーソルを移動し水平Ｉ字カーソルになったときにクリックし挿入横線を表示させて下さい。

[3.3.2] 2^{100} を計算して下さい。新しく生まれた2つのセルは `1.Addition` という見出しから延びるセル括弧でくくられていますね。

3.4　入力の省エネ化

Ｍａｔｈｅｍａｔｉｃａでは過去の入出力を再利用して同じ様なキーインを減らす色々な方法があります。ここでは直前の入出力の利用方法を学びます。

[3.4.1] 直前の出力セルの再利用法

- コントロール＋シフト＋L により`Out[2]` の値が次の行に太字でコピーされますね。
- そこで `^(1/100)` をキーインして実行して下さい。

[3.4.2] 直前の入力セルの再利用法

- コントロール＋L により 新しい行に `1267650600...376^(1/100)` が現れますね。
- そこの`100` の真ん中あたりにＩ字カーソルを持っていきダブルクリックして下さい。すると、`100` がハイライトされます。そこで 20 をキーインして下さい。すると`100` が `20` に置きかわりますね。そこで実行してみて下さい。

3.5　新しい節の作成

違った種類の作業に入るときは新しい節を作成しておくと作業の記録を管理し易くなります。

[3.5.1] `Out[3]` のセルの次に挿入横線を表示して下さい。

[3.5.2] `2.Helps` をキーインしセルを選択して ALT+4。この見出しは今までの節には組み込まれていませんね。

[3.5.3] 今までの節の最右セル括弧をダブルクリックして見出しだけにしておきましょう。

[3.5.4] これ以降の作業で生ずるセルはこのタイトル`2.Helps` というタイトルの節の下にまとめられます。

4　ヘルプ機能

4.1　関数名を知っているとき

[4.1.1] 見出し`2.Helps` の下に挿入横線を出して下さい。(下向き矢印のキーを押してもできます)

[4.1.2] `?Quotient` とキーインし実行して下さい。整数商を求める方法がわかります。

4.2　関数名の一部しか知らないとき

[4.2.1] `?Quotient` を再表示させて下さい (メニュー`Input/CopyInput from Above` でもできる)。`?Qu` 以外の字を消してから `*` をキーインして下さい。更に `?` の直後に I 字カーソルを置きクリックして下さい。点滅する縦棒があらわれますね。そこで `*` をキーインして下さい。

[4.2.2] `?*Qu*` と画面に表示されていることを確認してから実行して下さい。 4 つの関数名が表示されます。

4.3　関数名の一部しか知らないとき、補完機能の利用

知りたい関数の頭を知っているときは補完機能を利用できます。

[4.3.1] `?In` までキーインしてから コントロール+K を行って下さい。しばらくすると `In` から始まる関数の一覧表が現れますね。カーソルを動かすだけでスクロールされることに気付いて下さい。

[4.3.2] `t` をキーインすると`Int` から始まる関数だけに制限される。

[4.3.3] IntegerDigits が反転したときクリックしてください。すると `?In` が`?IntegerDigits` と補完されます。

[4.3.4] そこで実行して下さい。`IntegerDigits` の説明が現れます。

5 練習:有限力学系

具体的な数学的作業をやってみよう。

5.1 例題 1

$f1$ を整数係数の 1 変数多項式とし、集合 $X = \{ 0,1,2,...,9 \}$ 上の写像 f を $f(x) = f1(x)\mathrm{mod}10$ で定める。有限力学系 $\{ X,f \}$ のサイクルを求めよ。

5.2 有限力学系についての予備知識

5.2.1 定義

集合 X と写像 $\tau : X \to X$ の組 (X,τ) を離散力学系と呼ぶ。X を状態空間、X の元を状態、τ を状態遷移関数と呼ぶ。X が有限のとき、これを有限力学系という。

5.2.2 基本的概念

- $x \in X$ の軌道 $O(x)$ とは集合 $\left\{ \tau^i x \;\middle|\; i = 0,1,2,\cdots \right\}$.
- $x \in X$ が周期点であるとは、$\tau^p x = x$ を満たす $p > 0$ があることをいう。このような p の中で最小のものを x の周期 という。
- $\tau x = x$ を満たす x を固定点という。これは周期 1 の周期点である。
- $Y \subseteq X$ が部分力学系であるとは、$\tau Y \subseteq Y$ を 満たすものをいう。軌道は部分力学系である。周期点の軌道は部分力学系であるが、これをサイクルと呼ぶ。サイクル γ と交わる軌道は γ を含む。$O(x) \bigcap \gamma \neq \emptyset$ であるような x の全体を $B(\gamma)$ 書き、サイクル γ の鉢(basin) という。

5.3 演習

$f1(x) = x^2 + 1$ の場合にやってみよう。

5.3.1 関数の定義

- `Clear[f]` を実行して記号 f にまつわる過去を清算。
- `f[x_]:=Mod[x^2+1,10]` を実行。下線`_`を忘れないように！これで関数f が定義された。
- 試しに`f[3]` を実行して見よう。

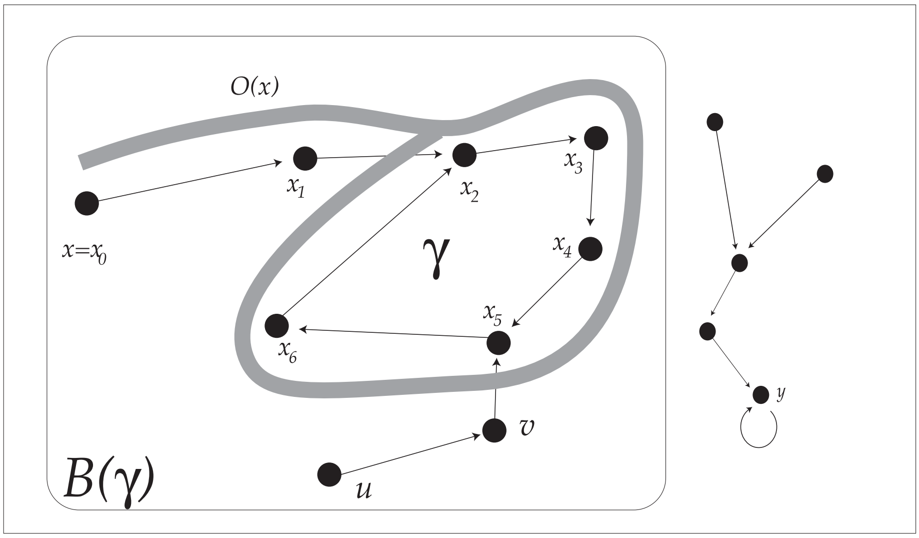

図 1: x を通る軌道．$x_k := \tau^k x$. x_2, x_3, x_4, x_5, x_6 は周期 5 の周期点．これらはサイクル γ を成す。x_0, x_1 は一過的である。y は固定点である。γ の鉢は γ に $\{ x_0, x_1, u, v \}$ を加えたものとなっている.

5.3.2　軌道を求める

- `NestList[f,1,20]` を実行。1 から始まる長さ２０の軌道がリストとして得られる。
- 練習　1 を他の値に変えて軌道を求めてみよ。(入力の再利用を使うと便利。)

5.3.3　リストによる集合の表現

次に集合 Y に対して $f(Y)$ を対応させる写像 ff を計算しよう。`Clear[u]` を実行してから、`u=Range[10]` を実行。これにより変数 u が集合Xを表すことになる。

5.3.4　手作業による ff の計算

- `Map[f,u]` を実行。これにより $\{ f[0], f[1], ..., f[9] \}$ というリストが計算される。
- `Union[%]` を実行。Union は重複する元を省く.

5.3.5　ff の定義

- `Clear[ff]; ff[y_]:=Union[Map[f,y]]` を実行。(`;` は出力を省くことを指示する)
- `ff[Range[10]]` を実行。前の結果と同じになっているか？

5.3.6　ff の軌道

`NestList[ff,Range[10],20]` を実行。

5.3.7　`f` のサイクル、即ち、`ff` の固定点の計算

`FixedPoint[ff,Range[10]]` を実行。§5.3.6 の結果と合っているか？

5.3.8　ff の固定点に至るまでの軌道の計算

`FixedPointList[ff,Range[10]]` を実行。§5.3.6 の結果と合っているか？

5.4　演習問題

[5.4.1] $f(x) = x^3 + x - 1\mathrm{mod}10$　のサイクルを求めよ。

[5.4.2] $\{\ 0, 1, ..., 99\ \}$ の上の力学系 $f(x) = x^2 + 1\mathrm{mod}100$　のサイクルを求めよ。

[5.4.3] $\{\ 0, ..., 999\ \}$ の上のランダム力学系 のサイクルを求めよ。ランダム力学系は `Do[ f[i] = Random[ Intege` を実行して定義できる。(`Clear[f]` を忘れないように！)

6 関数のグラフ表示

まず、一変数実関数のグラフ表示の方法を練習しよう。

6.1 基本 1

`Plot[fun,{x,xmin,xmax}]`

x の関数 fun を $[xmin, xmax]$ の範囲で表示する。

練習 :次を実行せよ。

[6.1.1] `Plot[x^2+x-1,{x,-1,1}]`
[6.1.2] `Plot[1/x,{x,-1,1}]`
[6.1.3] `Plot[Sin[x],{x,0,10 Pi}]`
[6.1.4] `Plot[Sin[x],{x,0,100 Pi}]`
[6.1.5] `Plot[Tan[x],{x,-2Pi,2Pi}]`
[6.1.6] `Plot[Sum[x^i/i!,{i,0,10}],{x,0,10}]`
($1 + x + x^2/2 + ... + x^{10}/10!$ のグラフ)
[6.1.7] `Plot[Product[(x-i),{i,0,10}],{x,0,10}]`
($x(x + 1)(x + 2)...(x + 10)$ のグラフ)

6.2 基本 2

`Plot[{fun1,fun2,..},{x,xmin,xmax}]`

x の関数 $fun1, fun2,..$ を $[xmin, xmax]$ の範囲で同時に表示する。

練習

[6.2.1] `Plot[{x,x^2,x^3,x^(1/2),x^(1/3),1/x,1/x^2},{x,-2,2}]`
[6.2.2] `Plot[{x,x^2,E^x,Log[x]},{x,0,1000}]`
[6.2.3]

```
Clear[f];
f[x_]=Product[(x-i),{i,0,3}];
Plot[{f[x],f'[x],f''[x]},{x,-1,4}]
```

6.3　表示の制御

関数についての適切で見やすい表示を得るためには、グラフの表示を色々制御すればよい。主な制御は次の点

- 表示する xy 平面の場所
- 縦軸横軸の単位の比
- 座標軸の表示
- 枠の表示
- 方眼の表示
- グラフの線のスタイル（太さ、色）

共通して次の形で指定：

Plot[f,{x,xmin,xmax}, opt,opt,...]

ただしオプション opt は `オプション名->値` という式。この指定は後で、別の指定をするまで効力を持つ。以下、個別に練習しよう。⋆は、デフォルトの設定。

6.4　表示範囲の指定

- 値の範囲を任せる：⋆`PlotRange -> Automatic`
- 値の範囲を $[ymin, ymax]$ に限る：`PlotRange -> {ymin,ymax}`

練習　：

[6.4.1] `Plot[Tan[x],{x,0,10Pi}]`
[6.4.2] `Show[%,PlotRange -> {-1,1}]`
[6.4.3] `Plot[{1/x,1/x^2,Log[x]},{x,-2,2},PlotRange -> {-1,1}]`

6.5　縦横の単位の比

- 単位長さの比を黄金比にする：⋆`AspectRatio -> automatic`
- （縦軸の単位長さ / 横軸の単位長さ）を a にする：`AspectRatio -> a`

練習　：

[6.5.1] `Plot[Sin[x],{x,-10,10}]`
[6.5.2] `Show[%,AspectRatio -> 1]`

[6.5.3] `Show[%,AspectRatio -> 10]`
[6.5.4] `Show[%,AspectRatio -> 0.1]`

6.6　座標軸の表示・指定

- 座標軸を描く：$\star$`Axes -> True`
- 座標軸を描かない： `Axes -> False`
- 軸の交点の位置は適当にする：$\star$`AxesOrigin -> Automatic`
  軸の交点の位置を (x,y) にする： `AxesOrigin -> {x,y}`
  軸に刻みを入れない： `Ticks -> None`
  軸に刻みを適当に入れる：$\star$`Ticks -> Automatic`
  刻みを入れる所の指定の例：`Ticks -> { points ,Automatic}`
  (points は実数のリストで、x 軸上のその元の所に刻みを入れる、たとえば`Range[0,10]` とすれば、0 から 10 までの整数の所に刻みが入る)

練習

[6.6.1] `Plot[1/(1+x^2),{x,-10,10},Axes -> False]`
[6.6.2] `Show[%,Axes -> True]`
[6.6.3] `Show[%,AxesOrigin -> {0,0}]`
[6.6.4] `Show[%,Ticks -> {Range[-10,10],Range[-10,10]}]`

6.7　枠の表示

枠を書かない*： `Frame -> False`
枠を書く： `Frame -> True`
枠に刻みを入れる*：`FrameTicks -> Automatic`
枠に刻みを入れない： `FrameTicks -> None`

練習

[6.7.1] `Plot[1/(1+x^2),{x,-10,10},Frame -> True]`
[6.7.2] `Show[%,FrameTicks -> None]`

6.8　方眼の表示

方眼を書かない*：`GridLines -> None`
方眼を書く： `GridLines -> Automatic`
y 軸に平行な線のみ書く：`GridLines -> {Automatic, None}`

x 軸の集合{x1,x2,..}を通って y 軸に平行な線のみ書く：
`GridLines -> {{x1,x2,..}, None}`

練習

```
(1) Plot[Sin[x],{x,0,10Pi},GridLines -> Automatic]
(2) Show[%,GridLines -> {Automatic,None}]
(3) Show[%,GridLines -> {None,Automatic}]
(4) Show[%,GridLines -> {Range[0,10,Pi],None}]
(5) Show[%,GridLines -> {None,Range[-1,1,0.1]}]
```

6.9　グラフ表示のスタイル

今までのオプションは他の平面グラフィックスにも使えるが、これは Plot にしか使えない。

色を指定：`PlotStyle -> RGBColor[r,g,b]`
`r,g,b` は `[0,1]` の範囲の実数で、赤・緑・青の割合を表す。

6.10　アニメ- ション機能

次にアニメーション機能を使う練習をします。いくつかの絵を描いてからそれを連続的に見せることでアニメーションを実現します。

【関数のグラフのアニメーション例】各絵を表示するスケールなどが同じでなければならないので、必ず同じ `PlotRange` を指定します。

```
Do[
        Plot[a Sin[x] ,{x,0,10Pi},PlotRange -> { -2,2}],
        {a,-1,1,0.1}
]
```

をまず実行してください。沢山のグラフ（２０枚）が描かれます。終ったらそれらのセルを覆っているセル括弧を選択して下さい。そして `Ctr+Y` を押して下さい。するとグラフが動きます。

`Do[仕事、範囲]` は、仕事を範囲で指定されているだけ行います。
範囲の指定法は：

`{i,n}`　：ｉを１からｎまで動かす
`{i,n0,n1}`　：ｉをｎ０からｎ１まで１刻みで動かす
`{i,n0,n1,di}`　：ｉをｎ０からｎ１まで　`di` 刻みで動かす

練習：１次関数族　`a x + a^2`　（a を適当に変える：例 `{a,-1,1,0.1}`）

【アニメーションの制御】スピード、方向、一コマずつ送る、などが可能です。
キーによる操作は：

スピード（上げる：>、下げる：<、１（最遅）－９（最速））
順番（順送り→、逆送り←、巡回 c、終了 q）
一コマずつ（後ろへ↑、前へ↓）

マウスによる操作はウィンドウのしたにある6つのボタンをクリックすることでもできます。どのボタンがどういう機能をするか、実験して見てください。

7 Mathematica の基本動作

7.1 Mathematica の式

練習 次を実行して式の正式な形を見よう：

```
2 // FullForm     (FullForm[2] と同じ)
1.0 // FullForm
f ’ //  FullForm
(x^2 + x y^5 ) ^ 3 // FullForm
a /. b -> x // FullForm
{a,b,c,d} // FullForm
```

練習 次を実行して式の木構造を見よう：

```
f ’ // TreeForm
(x^2 + x y^5 ) ^ 3 // TreeForm
a /. b -> x // TreeForm
{a,b,c,d} // TreeForm
f[g[g[x]],b,f[c,f[d,f[e,a]]]] // TreeForm
```

7.2 Mathematica の基本動作１：書き換え

練習 次の代入の働きを観察しなさい。

```
Clear[g,h,a,b,c];

g[a,b,a,g[a]] /. a -> 1
g[a,b,a,g[a]] /. a -> h[a]
2a /. a -> 5
```

数を文字へ書き換える

```
2a /. 2 -> b
```

式の書き換え

```
g[a,b,a,g[a]] /. g[a] -> g[h[a]]
```

式の頭部の書き換え

```
g[a,b,a,g[a]] /. g -> Plus
g[a,b,a,g[a]] /. g -> Times
```

代入の書き換え

```
2a /.a-> 1 /. a-> 2    (  ⇔    (2a /.a-> 1) /. a-> 2   )
2a /. ( a-> 1 /. a-> 2)
```

同時書き換え

```
g[a,b,a,g[a]] /. { a -> b, b -> a }
```

複数の書き換え

```
g[a,b,a,g[a]] /. { { a -> 1}, { a -> 2 } }
g[1,2,3,4] /. {{g -> Plus},{g-> Times }}
```

7.3 Mathematica の基本動作 2 ：割当　=

a=b　では、これを実行する時点での b の値が a に割り当てられる。

練習　次の割当の働きを予測してから実行して結果を観察しなさい。

```
Clear[g,h,a,b,x,y];

(1)            x=g[a,b,g[a]]
(2)            a=1
(3)            y=x ( ⇔  y=g[a,b,g[a]])
(4)            x

(5)            a=2
(6)            {x,y}

(7)            g[1]=b
(8)            {x,y}
(9)            z=y

(10)           g[1]=2
(11)           {x,y,z}

(12)           g=h[1]
(13)           {x,y,z}
```

7.4　Mathematica の基本動作 3 ：遅延割当　:=

`a:=b` では、`a` が使われる時点での `b` の値が `a` の代わりに代入される。

練習　次の遅延割当の働きを予測してから実行して結果を観察しなさい。

```
Clear[g,h,a,b,x,y,y1,z,z1];
(1)            x:=g[a,b,g[a]]

(2)            a=1
(3)            y:=g[a,b,g[a]]
        （ x と y とはまったく同じ働きをする）
(4)            y1=g[a,b,g[a]]
(5)             {x,y}

(6)            a=2
(7)            {x,y,y1}

(8)            g[1]=b
(9)            {x,y,y1}

(10)           z:=g[a,b,g[a]]
(11)           z1=g[a,b,g[a]]
(12)           {x,y,y1,z,z1}

(13)           g[1]=2
(14)           {x,y,y1,z,z1}

(15)           g=h[1]
(16)           {x,y,y1,z,z1}
```

7.5　書換と割当の関係

練習　結果を予測してから実行しなさい。

```
Clear[a,x];
a=5
2a /. a->3    （2a がまず計算される）
2a /. 10 -> x
2a /. 2 -> x
5x /. a -> 4   （⇔   5x /. 5 -> 4 ）
```

8　プログラミング

Mathematica には、関数・命令が豊富に用意されてはいますが、自分の目的にピッタリのものはないのが普通です。しかし、比較的簡単なプログラムにより、自分の作業に必要なものを作ることが出来ます。きょうは、Mathematica のプログラミングの中で、最も基本的なこと：

- 関数定義
- 繰り返し
- 条件分岐

を学びます。

8.1　関数定義（「マクロ」として）: １行プログラム

繰り返し行う作業が長い入力を要するときに名前を付けると便利である（この名前をマクロと呼ぶことがある）。

8.1.1　例

`Sin[x^3+a x^2 + 1]` のグラフを色々な値の a について描かせたいとき、

（＊）　`a=1;Plot[Sin[x^3+a x^2 +1],{x,0,10}]`

を実行し、a を変える毎に下線部をもう一度実行すればよい。そのためには、
以前学んだ CTR + L で直前の入力を再表示されればよいが、今回はもっと発展性
のある方法を学ぶ。それは下線部に名前を付けることである：

```
plotSin:=Plot[Sin[x^3+a x^2 +1],{x,0,10}]
```

（等号の前の：を忘れないように！）を実行しておくと、

```
a=1;plotSin
```

によって、（＊）と同じことを行える。

注意：
◇ 名前としては英字から始まる任意の英数字列を使うことが出来る。
◇ パスカルやＣとは違って下線（＿）は名前の一部として使えない。
◇ 組み込み関数は大文字から始まるので、自分で定義する関数は小文字から始まるようにしたほうが良い。

8.2 引数を持つ関数定義

上のように、パラメータを含む作業の時は、そのパラメータをマクロの引数として与えることが出来る。

```
plotSin1[a_]:=Plot[Sin[x^3+a x^2 +1],{x,0,10}]
```

（aの後の下線を忘れないように）

を実行しておくと、今度は単に、

```
plotSin1[1]
```

によって、（＊）と同じことを行える。

8.2.1 例

```
plotF2[f_,a_]:=Plot[f[x],{x,-a,a}]
```

を実行すれば、`plotF2[Sin,Pi]` によって `Sin` のグラフを `[-Pi,Pi]` の範囲で書ける。

8.3 練習

次の作業を容易にするマクロを定義しなさい：

（１）`x^n-1` の因数分解（`Factor`）を色々な `n` について行う。

（２）曲線（`Sin[a*x],Cos[b*x]`）を色々な `a,b` について表示する。
（`ParametricPlot`）

8.4　関数定義 (「マクロ」として): 複数行プログラム

繰り返し行う作業が、数回の入力- 実行から成る場合には、`Module` を用いる :

`Module[{ 局所変数，...，局所変数}，実行文; 実行文; 実行文; 実行文]`

値は最後の文の値となる。

8.4.1　例

```
exCoef[n_,i_]:= Module[ {u},
                  u = Expand[ Product[x+i,{i,n}]];
                  Coefficient[ u, x^i]
                     ]
```

注意
◇ これを

```
exCoef2[n_,i_]:=Coefficient[Expand[ Product[x+i,{i,n}]],x^i]
```

としても良いが、わかりにくい。関数の入れ子はなるべく避けて、補助変数を用いて (少々長くなっても) わかりやすく書くことがプログラミングの能率を上げる。

◇ 改行やタブを利用して、関数と引数との関係が一目瞭然となるようにすると誤りを減らし、誤りの修正を容易にすることが出来る。

◇ 中間結果を表示させたいときは、`Print` を用いる。

8.5 繰り返し：Ｄｏ文

同種の作業をパラメータを変えながら繰り返して行うには、何通りかの方法がある。最も単純なＤｏ文を今日は覚えよう：

`Do[やること、パラメータの範囲]`

パラメータの範囲としは次のような指定が可能：

{数ｎ}: やることをｎ回繰り返す
{パラメータ名ｉ, 数ｎ}: ｉを１からｎまで動かす
{パラメータ名ｉ, 数ｎ1, 数ｎ2}: ｉをｎ１からｎ２まで動かす
{パラメータ名ｉ, 数ｎ1, 数ｎ2, 数ｄｎ}: ｉをｎ１からｎ２迄ｄｎ刻みで動かす

注意：

◇ Do 文で動かすパラメータが他にも使われているときは予想外の影響を及ぼす危険があるので、`Module` を使った方がよい：

例：`Module[{i}, Do[...,{i,10}]]`

◇ Do 文の各ステップで何かを出力させたい時は、`Module` の場合と同じで「`Print`」を用いる：

例：`Do[Print[a^2],{a,1,10,0.1}]`

8.6 条件分岐：Ｉｆ文

`If`［条件Ｐ，実行文Ｘ１］

は、条件ＰがTrueならばＸ１の値をとる。（ＰがTrueでなければNullという値になる。）

`If`［条件Ｐ，実行文Ｘ１，実行文Ｘ２］

は、条件ＰがTrueならばＸ１、FalseならばＸ２の値をとる。

`If`［条件Ｐ，実行文Ｘ１，実行文Ｘ２，実行文Ｘ３］

は、条件ＰがTrueならばＸ１、FalseならばＸ２、どちらでもなければＸ３の値をとる。

条件の形式：（a,bが式、p,q,rが論理式のとき）

`a==b`	aとbの値が同じ時True，そうでないときFalse， aまたはbが値を持たないときはa==bが値。
`a!=b`	a==bの否定
`a>=b`	aはb以上
`a<=b`	aはb以下
`!p`	pの否定
`p && q && r`	p かつ ｑかつｒ
`p || q || r`	p または ｑまたはｒ

論理関数：

`IntegerQ,MatrixQ,Negative,NonNegative,NumberQ,OddQ,Positive,PrimeQ,VectorQ`

例：ステップ関数

```
f[x_]:= If[x<0 || x > 1,0,1]
```

8.6.1 練習

ａ：偶数のとき２でわり、奇数のときは３倍して１を加える関数を定義しなさい。
ｂ：整数部分が偶数のとき１、奇数のときは０をなる関数を定義しなさい。
ｃ：１万までの素数を表示しなさい。（素数判定をするPrimeQを利用しなさい。）

8.7 場合分けによる関数定義

定義式の右に $\boxed{\texttt{/;}\quad \text{条件式P}}$ をつけ加えると、Pが成り立つ時だけ定義式が適用される。

例 Collaz の力学系。

```
f1[x_]:= x/2 /; EvenQ[x]
f1[x_]:= 3x + 1 /; OddQ[x]
```

練習問題 偶数区間で $x-[x]$、奇数区間で $-x+[x]$ となる関数を定義し、グラフを書きなさい。

8.8 条件

`Which[cond_1,body_1,cond_2,body_2,...,cond_n,body_n]`
　　`cond_i` が成り立つとき `body_i` が値となる
`Switch[expr,form_1,body_1,form_2,body_2,...,form_n,body_n]`
　　式 `expr` が `form_i` であるとき `body_i` が値となる。

例：

```
f2[x_]:=Which[x<0,0,x>=1,0,True,1]
Plot[f2[t],{t,-1,2}]

f3[x_Integer]:=Switch[Mod[x,3],0,a,1,b,2,c]
Table[f3[a],{a,0,10,0.5}]

f4[x_]:=Switch[x,0,a,1,b,u_+v_,v]
        f4[x^2+y]
        f4[x y]
```

8.9 繰り返し：While

```
While[cond,body]
```

条件 cond が成り立っている間 body を実行する

例：

```
x=0;n=1;While[n<15,   x+=n; n++]
```

（While[n<15, x+=n; n++] は While[n<15, x=x+n; n=n+1] と同じ）

略記法（C でも常用する便利な書き方なので覚えよう）

`f[x=g]` ⇔ `x=g; f[x]`
`f[i++]` ⇔ `f[i]; i=i+1`
`f[++i]` ⇔ `i=i+1; f[i]`
`f[i--]` ⇔ `i=i-1;  f[i]`
`i+=di` ⇔ `i=i+di`
`i-=di` ⇔ `i=i-di`
`x *= c` ⇔ `x=x*c`
`x /= c` ⇔ `x=x/c`

練習

Do[Print[i], {i,10}] を While を用いて書きなさい。

8.10 繰り返し：For

```
For[  start,  cond,  incr,  body  ]
```

（１）初期設定 start を行う。
（２）body を実行し、パラメータの変更 incr を行う。
（３）条件 cond が成り立っているならば（２）を行う。

例：

```
For[x=0;n=1, n<15, n++, x+=n]
```

練習　Do[Print[i],{i,10}] を For を用いて書きなさい。

Mathematica は式のパターンを指定する便利な書式を持っています。これを用いると、式をきめ細かく書換えることができます。従って、関数定義の自由度も飛躍的に増えます。

8.11　パターンの基本例１：f[n_]

`f[n_]`　頭部が `f` である式で、引数を `n` という名前で引用する

使用例

（1）　`{f[1],f[2],f[t]} /. f[n_] -> n^2`

（2）　`g[Point[p_]]:=p`
　　　`g[Point[{1,2}]]`

同様の例：

`f[n_,m_]`　頭部が `f` で２引数を持つ式で、引数を `n,m` という名前で引用する

`f[n_,n_]`　頭部が `f` である式で、２引数が一致する場合を、その引数を `n` という名前で引用する

`f[n,m_]` 頭部が `f` で２引数を持ち , 第１引数が `n` である式。第２引数を `m` という名前で引用する。

これらの特別なものとして次がある：

`x^n_`　⇔　`Power[x,n_]`　`x` のべきで、指数を `n` という名前で引用する

`x_^n_`　⇔　べきの形をした式で、底を `x`, 指数を `n` という名前で引用する

`a_+b_`　⇔　`Plus[a_,b_]`　和の形をした式

`{a1_,a2_}`　⇔　`List[a1_,a2_]`　２成分を持つリストで、成分を各々`a1,a2` という名前で引用する

使用例：

```
f5[x_,1-x_]:=f6[x]
f5[x^2,1-x^2]
f5[4,-3]
```

8.12 パターンの基本例２：x_h

`x_h` 頭部が `h` の式で、それ自身を `x` という名前で引用する

これの特別なものとして次がある：

```
x_Integer
x_Real
x_Complex
x_List
```

使用例：

```
f7[x_Integer]:= Table[Binomial[x,i],{i,0,x}]]
f7[x_List]:= First[x]
f7[ddd[x_]]:=Print[x]

f7[3]
f7[{3.5,2}]
f7[ddd["Hallo!"]]
f7[3.5]
```

8.13 パターンの基本例３：x_?test

`x_?test` `test` を満たす式で、それ自身を `x` という名前で引用する

使用例：

```
f2[x_?EvenQ]:= x/2
f2[x_?OddQ]:= 3x-1
```

8.14 パターンの基本例４：x___

`x__`（xの後に下線_が２つ）: 長さ１以上の式の列で、xという名前で引用する。（名前が要らなければ単に：__）

`x___` （xの後に下線_が３つ）: 長さが０以上の式の列で、xという名前で引用する。（名前が要らなければ単に：___）

使用例：長さが２以上のリストについては最初の２成分を取り出し、長さが１のリストは成分を重複させたリストを与える。

```
f8[{x_,y_,z___}]:={{x,y},z}
f8[{x_}]:={x,x}
f8[{1}]
f8[{1,2,3}]
f8[{1,2,3,4,5}]
```

8.15 パターンの基本例５：x_/;cond

`x_/;cond` 条件 cond を満たす式で、xという名前で引用する。

使用例：

```
f9[x_/; x<0 || x>+1]:=0;
f9[x_/; 0<x && x<1]:=1;
Plot[f9[t],{t,-1,2}]
```

注意：上の定義は

```
f10[x_/; x<0]:=0;      （１）
f10[x_/; x<1]:=1;      （２）
f10[x_]:=0;
```

としても良い。複数の定義式があるとき、`Mathematica` は上から順に適用を試みるからである。例えば、`f10[-1]` が評価されるとき、（１）が適用できるので（２）の適用が試みられることはない。